半正定値計画問題に対する内点法ソフトウェアSDPA (SemiDefinite Programming Algorithm) (最適化のための連続と離散数理)

AUTHOR(S):

藤澤, 克樹; 小島, 政和; 中田, 和秀

数理解析研究所講究録
1114巻 1999年 149-159

半正定値計画問題に対する内点法ソフトウェア SDPA(SemiDefinite Programming Algorithm)

藤沢 克樹 (Katsuki Fujisawa)[1]
小島 政和 (Masakazu Kojima)[2]
中田 和秀 (Kazuhide Nakata)[3]

1 はじめに

最適化手法として頻繁に用いられる数理計画法の中から, 近年研究の進展が特に目覚しく, その幅広い応用を期待されている半正定値計画問題 (Semidefinite Programming : SDP) を取り上げる. SDP などの最適化手法が最近特に注目を集めている理由には以下のようなものが考えられる. 例えば建築構造物や航空機などの構造設計においては, 単にデザイン的, 機能的に複雑な要求を満たす設計を見出すだけでなく, できるだけ速く, 安く完成させることも非常に重要である. そのためには前もって様々な要求を制約条件として記述して最も効率の良い解を求めていく最適設計を行なっていくことが好ましいが, 従来は大規模な構造物に対して最適設計を行なうことは, 最適化手法や計算機の能力からみても非常に困難であった. しかし最近の (SDP などの) 最適化手法と計算機環境の著しい進歩によって最適設計を目指す試みは加速され, 現実味を帯びはじめている.

SDP は線形計画問題 (LP) や凸二次計画問題などを含んだより大きな凸計画問題の枠組であるが, 制約に半正定値制約という非線形制約を持っている. 従って SDP として定式化できる最適化問題が解けるだけでなく, 非凸最適化問題に対する強力な緩和値を導き出すことができる. そのため SDP を繰り返して解くことによって (最適に解くことが極めて難しいが非常に実用上重要な) 非凸最適化問題を扱える可能性を持っている [1]. また, 著者らが開発したソフトウェア SDPA (SemiDefinite Programming Algorithm) [2][1] をはじめとして, 複数の研究グループによって SDP に対するソフトウェアが開発され, インターネットより公開されている. ここ数年の間に多くの実験的解析が行われると共に, それらの結果をフィードバックすることにより SDP のアルゴリズム自体も進歩を遂げた [3].

最適設計を目指し, さらに複雑で大規模な問題を解くためには, 理論的成果を随時組み入れると共に, 最新のコンピュータ技術 (並列, 広域計算) 等との融合も必要不可欠であって [4],SDPA などの最適化手法を組み込んだ広域並列計算システムは現在開発中である. 本解説では SDP と SDP に対する主双対内点法について触れたあとに SDPA の実装方法や実験結果について述べる.

[1]京都大学大学院 工学研究科 建築学専攻 助手
[2]東京工業大学大学院 情報理工学研究科 数理・計算科学専攻 教授
[2]東京大学大学院 工学系研究科 物理工学専攻 助手
[1]ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA/

2 半正定値計画問題 (Semidefinite Programming : SDP)

SDP に対するサーベイには Vandenberghe と Boyd [5] などがある. また SDP に関する文献やソフトウェアなどの情報を集めたホームページも存在するので参考にしていただきたい[2][3]. SDP が期待されている主な工学的分野には構造最適化 [6], システムと制御 [7], 組合せ最適化 [8] などがある.

次に SDP に関する諸定義を行なう. $\Re^{n\times n}$ を $n\times n$ の実行列の集合, $\mathcal{S}^n$ を $n\times n$ の実対称行列の集合とする. 任意の $X, Z \in \Re^{n\times n}$ に対して, $X \bullet Z$ は X と Z の内積, すなわち, Tr X^TZ (X^TZ の trace : 固有和) を表す. $X \succ O$ は $X \in \mathcal{S}^n$ が正定値, つまり任意の $u(\neq 0) \in \Re^n$ に対し $u^TXu > 0$ であることを示している. また $X \succeq O$ は $X \in \mathcal{S}^n$ が半正定値, つまり任意の $u \in \Re^n$ に対し $u^TXu \geq 0$ であることを示している.

$C \in \mathcal{S}^n$, $A_i \in \mathcal{S}^n$ $(1 \leq i \leq m)$, $b_i, y_i \in \Re$ $(1 \leq i \leq m)$, $X \in \mathcal{S}^n, Z \in \mathcal{S}^n$ とする. このとき, SDP の主問題と双対問題は以下のように与えられる.

$$
\left.
\begin{array}{ll}
\textbf{主問題：} & \\
\text{最小化} & C \bullet X \\
\text{制約条件} & A_i \bullet X = b_i \ (1 \leq i \leq m), \\
 & X \succeq O. \\
\textbf{双対問題：} & \\
\text{最大化} & \displaystyle\sum_{i=1}^{m} b_i y_i \\
\text{制約条件} & \displaystyle\sum_{i=1}^{m} A_i y_i + Z = C, \\
 & Z \succeq O.
\end{array}
\right\} \tag{1}
$$

本解説を通して, $A_i \in \mathcal{S}^n (1 \leq i \leq m)$ が線形独立であることを仮定する. (X, y, Z) が SDP の実行可能解であるとは, X が主問題の実行可能解であり、(y, Z) が双対問題の実行可能解であることを表す. また, (X, y, Z) が SDP の実行可能内点解であるとは, X が主問題の実行可能内点解 (つまり, $X \succ O$ を満たす実行可能解) であり, (y, Z) が双対問題の実行可能内点解 (つまり, $Z \succ O$ を満たす実行可能解) の場合である.

2.1 SDP に対する主双対内点法

以下では, SDP の主双対内点法の枠組みについて簡単に述べる. SDP (1) には内点実行可能解が存在することを仮定する. このとき双対定理により, SDP (1) の最適解は以下の式を満たす.

$$
\left.
\begin{array}{l}
A_i \bullet X = b_i \ (1 \leq i \leq m), \\
\displaystyle\sum_{i=1}^{m} A_i y_i + Z = C, \\
X \succeq O, \ Z \succeq O, \\
XZ = O.
\end{array}
\right\} \tag{2}
$$

[2] http://new-rutcor.rutgers.edu/~alizadeh/sdp.html
[3] http://www.zib.de/helmberg/semidef.html

SDP に対する主双対内点法は線形計画問題に対する主双対内点法と同様に, 中心パス $\mathcal{P}$ に沿って進みながら最適解に収束する反復解法である. ここで、中心パス $\mathcal{P}$ はパラメータ μ を用いて以下のように定義される.

$$\mathcal{P} = \left\{ (X,y,Z) \left| \begin{array}{l} A_i \bullet X = b_i \\ (1 \le i \le m), \\ \sum_{i=1}^{m} A_i y_i + Z = C, \\ X \succ O,\ Z \succ O, \\ XZ = \mu I \ (\mu > 0). \end{array} \right. \right\}. \tag{3}$$

ただし $I \in \mathcal{S}^n$ は単位行列を表す. 任意の $\mu > 0$ に対し, 中心パス $\mathcal{P}$ 上の点が一意に存在することが知られている. また, 中心パス $\mathcal{P}$ は SDP の内点実行可能解の集合の内部の滑らかな曲線となっており $\mu \to 0$ のとき SDP の最適解に収束することも知られている [9].

SDP の主双対内点法では, 探索方向パラメータ $\mu > 0$ を選び下記の条件を満たす中心パス上の点 (X,y,Z) をターゲットとし, その Newton 方向に進む.

$$\left. \begin{array}{l} A_i \bullet X = b_i \ (1 \le i \le m), \\ \sum_{i=1}^{m} A_i y_i + Z = C, \\ XZ = \mu I \end{array} \right\} \tag{4}$$

次に SDP に対する一般的な主双対内点法の概要を以下に示す. なお Mehrotra 型 [13] の主双対内点法が存在するが, Mehrotra 型 は説明が複雑になるのでこちらは SDPA [3] を参考にしていただきたい.

SDP に対する主双対内点法

手順0. $k = 0$ として, $X^0 \succ O$, $Z^0 \succ O$ を満たす初期解 (X^0, y^0, Z^0) を選ぶ (実行可能解でなくてもよい).

手順1. 現在の解 (X^k, y^k, Z^k) が終了条件を満たすならば, アルゴリズムを終了する.

手順2. 以下のような条件を満たすような探索方向 (dX^k, dy^k, dZ^k) を計算する.

$$(X^k + dX^k, y^k + dy^k, Z^k + dZ^k) \in \mathcal{P}$$

具体的には探索方向を計算するために次の Newton 方程式を解く.

$$\begin{array}{l} A_i \bullet (X^k + dX^k) = a_i \ (i = 1, 2, \ldots, m), \\ (Z^k + dZ^k) = C - \sum_{i=1}^{m} A_i (y_i^k + dy_i^k), \\ X^k Z^k + X^k dZ^k + dX^k Z^k = \mu I, \\ dX^k \in \mathcal{S}^n,\ dZ^k \in \mathcal{S}^n,\ dy^k \in R^m. \end{array}$$

手順3. 新しい反復点 $(X^{k+1}, y^{k+1}, Z^{k+1})$ を次の条件

$$\begin{array}{l} X^{k+1} = X^k + \alpha_p dX^k \succ O, \\ Z^{k+1} = Z^k + \alpha_d dZ^k \succ O, \end{array}$$

を満たすように定める.

手順4. $k=k+1$ として, 手順1に戻る.

ただし, $\alpha_p, \alpha_d > 0$ はステップ長. (ステップ長をあまり大きく取ると新しい反復点が実行可能領域の境界に近づきすぎて, 以後の反復で困難を生ずる. また, 小さすぎると収束までに多くの反復回数を要する).

2.2 探索方向の計算方法

一般に (4) において XZ は対称行列でないので, 直接 Newton 法を適用できない. このため, 現在までに様々な探索方向 $(dX, d\boldsymbol{y}, dZ)$ が提案されてきた. 実用上有用な探索方向としては, HRVW/KSH/M 方向 [9–11] と NT 方向 [12] などが知られている.

以下では, $p_i = b_i - A_i \bullet X$, $D = C - \sum_{i=1}^{m} A_i y_i - Z$, $K = \mu I - XZ$ とおく.

HRVW/KSH/M 方向: HRVW/KSH/M 方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ は以下のようにして求めることができる.

$$\left.\begin{aligned} &B d\boldsymbol{y} = s,\ dZ = D - \sum_{j=1}^{m} A_j d\boldsymbol{y}_j, \\ &\widehat{dX} = X(X^{-1}K - d\boldsymbol{Z})Z^{-1}, \\ &dX = (\widehat{dX} + \widehat{dX}^T)/2. \end{aligned}\right\} \tag{5}$$

ただし,

$$\begin{aligned} &B_{ij} = XA_iZ^{-1} \bullet A_j \ (1 \le i \le m,\ 1 \le j \le m), \\ &s_i = p_i - A_i \bullet X(X^{-1}K - D)Z^{-1},\ (1 \le i \le m). \end{aligned}$$

ここで $n \times n$ 行列の X, $n \times n$ 行列の Z^{-1} さらに $m \times m$ 行列の B は全て対称行列であり, 一般的には A_i $(1 \le i \le m)$ の全てが疎行列であったとしても X, Z^{-1}, B は全て密行列になる. そのため係数行列 B および右辺項 s を計算するにはそれぞれ $O(mn^3 + m^2n^2)$, $O(n^3 + mn^2)$ の計算量が必要になる. 一方 (5) を解いて探索方向 $(dX, d\boldsymbol{y}, dZ)$ を求めるにはコレスキー分解 (Cholesky factorization) などを使用して $O(m^3 + n^3)$ の計算量が必要になる. それゆえ係数行列 B を計算する部分は、s の計算や線形方程式の計算 $Bd\boldsymbol{y} = s$ よりも HRVW/KSH/M 方向の計算時間全体の中で大きな割合を占める. それゆえ,A_i $(1 \le i \le m)$ が疎行列である場合には, その特質を活用して B の計算量を下げることが全体の計算量の軽減に大きく貢献することになる.

NT 方向: 行列 W を

$$W = X^{1/2}(X^{1/2}ZX^{1/2})^{-1/2}X^{1/2} \tag{6}$$

$$= Z^{-1/2}(Z^{1/2}XZ^{1/2})^{1/2}Z^{-1/2}, \tag{7}$$

このとき,NT 方向 $(dX, d\boldsymbol{y}, dZ)$ は以下のようにして求めることができる.

$$\left.\begin{aligned} &B d\boldsymbol{y} = s,\ dZ = D - \sum_{j=1}^{m} A_j d\boldsymbol{y}_j, \\ &\widehat{dX} = W(X^{-1}K - dZ)W, \\ &dX = (\widehat{dX} + \widehat{dX}^T)/2. \end{aligned}\right\} \tag{8}$$

ここで

$$B_{ij} = WA_iW \bullet A_j \ (1 \leq i \leq m,\ 1 \leq j \leq m),$$
$$s_i = p_i - A_i \bullet W(X^{-1}K - D)W \ (1 \leq i \leq m).$$

(6,7) より行列 W を求めた後は、NT 方向の計算方法は HRVW/KSH/M 方向の計算方法とほぼ同等であり、計算量やメモリ使用量はほとんど変わらない. 従って次節で解説するように 2 つの探索方向の計算方法に対して, 同様の工夫が有効になる.

3 SDPA の実装方法

この節では, 筆者らが作成した SDP に対する主双対内点法ソフトウェア SDPA [2] の実装方法について説明を行なう.

3.1 データ構造 (ブロック対角行列)

SDPA ではブロック対角な行列のデータ構造およびそれらの内部演算を組み入れている. 次にブロック対角行列の一般形を示す. ブロック数とブロック構造ベクトルを用いて, 定数行列 A_i や 変数行列 X, Z に共通なブロックデータ構造を表現する.

$$\left.\begin{array}{l} F = \begin{pmatrix} G_1 & O & O & \cdots & O \\ O & G_2 & O & \cdots & O \\ \cdot & \cdot & \cdot & \cdots & O \\ O & O & O & \cdots & G_\ell \end{pmatrix}, \\ G_i:\ p_i \times p_i \text{ 対称行列 } (i = 1, 2, \ldots, \ell). \end{array}\right\}$$

一般的には, ある行列を入力したときにその行列のブロック対角構造を自動的に判別するのは時間がかかるので SDPA では以下のように nBLOCK や bLOCKsTRUCT などのパラメータを用いてユーザーがブロック対角構造を指定するようになっている.

$$\text{nBLOCK} = \ell,$$
$$\text{bLOCKsTRUCT} = (\beta_1, \beta_2, \ldots, \beta_\ell),$$
$$\beta_i = \begin{cases} p_i : G_i \text{ が通常の対称行列のとき} \\ -p_i : G_i \text{ が対角行列のとき} \end{cases}$$

3.2 疎行列を利用した HRVW/KSH/M と NT 方向の計算法

この節では,SDPA が現在組み込んでいる 2 つの探索方向 (HRVW/KSH/M, NT) を効率良く計算する方法を紹介する [14]. 特に HRVW/KSH/M と NT 方向に関しては定数行列 A_i $(1 \leq i \leq m)$ の全てまたは幾つかが疎行列であるときに効率的にそれらの方向を計算することができる. しかし A_i の全てが密行列であっても通常の方法と比較して効率が悪くなるわけではない. この計算方法の特徴を簡単にまとめると以下のようになる.

1. SDPA は A_i が密 (dense)、疎 (sparse) の場合に備えて, それぞれ専用の計算方法を用意している.
2. SDPA は A_i の非零要素の数から自動的に計算方法を判断する.
3. 例えば G_0 は密行列、G_1 は疎行列といったような場合には, 各ブロックごとに異なる計算方法を自動的に選択することができる.

2.1 節で述べたように SDPA の 1 反復の中では $m \times m$ の対称行列である B の計算が通常では最も多くの実行時間を占める.

次に 2 つの探索方向に対する計算方法を同時に考慮するために,

$$B_{ij} = TA_iU \bullet A_j \tag{9}$$

$(1 \leq i \leq m,\ 1 \leq j \leq m)$ を導入する. このとき B_{ij} は行列 B の (i,j) 成分, また $T \in \mathcal{S}^n$, $U \in \mathcal{S}^n$ である. HRVW/KSH/M 方向の場合では $T = X, U = Z^{-1}$ とすればよく. NT 方向の場合では $T = U = W$ とすればよい. B は対称行列なので, 行列 B の上三角部分, つまり B_{ij} $(1 \leq i \leq j \leq m)$ のみ考慮すればよい.

行列 B の実際の計算時間には浮動小数点の乗算や加算, また行列 A_i $(1 \leq i \leq m)$ の各要素をメモリから取り出すための時間など多くの要因が含まれている. この場合加算の回数は乗算の回数とほぼ同じオーダーであるが (一般的には浮動小数点の乗算は加算よりも多くの計算時間を必要とする), 一方, 行列 A_i から要素を取り出す時間は、データ構造の選択に大きく影響される. そこで, 最初に乗算の回数に実行時間の解析の焦点をあてて, その後 A_i $(1 \leq i \leq m)$ の各要素をメモリから取り出す時間 (いわゆるオーバーヘッド) も考慮に入れて行くことにする.

ここで,f_i を A_i の非零要素の数とする. また $\mathbf{\Sigma}$ は索引 (index) $1, 2, \ldots, m$ の順列の集合とする. (つまり, $\{1, 2, \ldots, m\}$ からそれ自身への一対一写像である). $\boldsymbol{\sigma} \in \mathbf{\Sigma}$ は次のように B_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq m)$ 要素を計算する順序を決定する.

$$\left.\begin{array}{l} \overrightarrow{B_{\sigma(1)\sigma(1)}, B_{\sigma(1)\sigma(2)}, \ldots B_{\sigma(1)\sigma(m)},} \\ \qquad \overrightarrow{B_{\sigma(2)\sigma(2)}, \ldots B_{\sigma(2)\sigma(m)},} \\ \qquad\qquad \overrightarrow{\cdots \cdots} \\ \qquad\qquad\qquad B_{\sigma(m)\sigma(m)}. \end{array}\right\} \tag{10}$$

既に述べたように B は対称行列なので $B_{\sigma(j)\sigma(i)} = B_{\sigma(i)\sigma(j)}$ $(1 \leq i < j \leq m)$ である.

次に $\boldsymbol{\sigma} \in \mathbf{\Sigma}$ と $i \in \{1, 2, \ldots, m\}$ を固定し, $B_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算する 3 つの方法を提案する.

$\mathcal{F}$-1: 始めに $F_i = A_{\sigma(i)}U$ ($nf_{\sigma(i)}$ 回の乗算を必要とする) を計算して, 次に $M_i = TF_i$ (n^3 回の乗算が必要) の計算を行う. その後各 $j = i, i+1, \ldots, m$ に対して $B_{\sigma(i)\sigma(j)} = M_i \bullet A_{\sigma(j)}$ ($f_{\sigma(j)}$ 回の乗算が必要) の計算を行う. よって, 全ての $B_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の回数の合計は次の式から得ることができる.

$$nf_{\sigma(i)} + n^3 + \sum_{i \leq j \leq m} f_{\sigma(j)}. \tag{11}$$

$\mathcal{F}$-2: 始めに $F_i = A_{\sigma(i)}U$ ($nf_{\sigma(i)}$ 回の乗算が必要) の計算を行う. 次に各 $j = i, i+1, \ldots, m$ に対して,

$$B_{\sigma(i)\sigma(j)} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[A_{\sigma(j)}]_{\alpha\beta}\left(\sum_{\gamma=1}^{n}T_{\alpha\gamma}[F_i]_{\gamma\beta}\right),$$

の計算を行う ($(n+1)f_{\sigma(j)}$ 回の乗算が必要). よって, 全ての $B_{\sigma(i)\sigma(j)}$ $(i \le j \le m)$ を計算するための乗算の回数の合計は次の式から得ることができる.

$$nf_{\sigma(i)} + (n+1)\sum_{i\le j\le m} f_{\sigma(j)}. \tag{12}$$

$\mathcal{F}$-3: 各 $j = i, i+1, \ldots, m$ に対して,

$$B_{\sigma(i)\sigma(j)} = \sum_{\gamma=1}^{n}\sum_{\epsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[A_{\sigma(i)}]_{\alpha\beta}T_{\alpha\gamma}U_{\beta\epsilon}\right)[A_{\sigma(j)}]_{\gamma\epsilon},$$

を計算する ($(2f_{\sigma(i)}+1)f_{\sigma(j)}$ 回の乗算が必要). よって, 全ての $B_{\sigma(i)\sigma(j)}$ $(i \le j \le m)$ を計算するための乗算の回数の合計は次の式から得ることができる.

$$(2f_{\sigma(i)}+1)\sum_{i\le j\le m} f_{\sigma(j)} \tag{13}$$

概念的には, 行列 A_i, A_j が共に密な場合には $\mathcal{F}$-1, 共に疎な場合には $\mathcal{F}$-3, 一方が密でもう一方が疎な場合には $\mathcal{F}$-2 を用いて B_{ij} の計算を行なう.

次に行列 A_i $(1 \le i \le m)$ の要素を取り出すための時間 (オーバーヘッド) を考慮に入れることにする. 最初に次の仮定を置く.

(i) 全ての A_i $(1 \le i \le m)$ は 疎行列用のデータ構造で保持するものとする.

(ii) 行列 T と U の B_i 部分は通常の二次元配列で保持する (多くの場合密であるから).

(iii) 疎行列用のデータ構造から要素を取り出す操作の方が、通常の二次元配列から要素を取り出すよりも実行時間がかかるものとする.

これらの仮定より, (11), (12), (13) 式に対して, 以下のように修正を行う. ここで式 $\mathcal{F}$-k $(k = 1, 2, 3)$ を用いて, 全ての $B_{\sigma(i)\sigma(j)}$ $(i \le j \le m)$ を計算するための乗算回数の重み付き合計数 $d_{ki}(\boldsymbol{\sigma})$ を定義する.

$$d_{1i}(\boldsymbol{\sigma}) = \kappa n f_{\sigma(i)} + n^3 + \kappa\sum_{i\le j\le m} f_{\sigma(j)} \tag{11}'$$

$$d_{2i}(\boldsymbol{\sigma}) = \kappa n f_{\sigma(i)} + \kappa(n+1)\sum_{i\le j\le m} f_{\sigma(j)}, \tag{12}'$$

$$d_{3i}(\boldsymbol{\sigma}) = \kappa(2\kappa f_{\sigma(i)}+1)\sum_{i\le j\le m} f_{\sigma(j)}. \tag{13}'$$

ここで $\kappa \geq 1$ は定数であり, κ を疎行列データ構造を持つ A_i $(1 \leq i \leq m)$ の各要素を取り出すためのオーバーヘッドとみなすことにする. もし $\kappa = 1$ ならば (11)', (12)', (13)' は修正前の定義である (11), (12), (13) と一致する. SDPA においては, ある A_i $(1 \leq i \leq m)$ が与えられたとき, 計算方法 $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の実際のパフォーマンスを推定するための指標として $d_{ki}(\boldsymbol{\sigma})$ を使用している. κ の値の適正値は採用するデータ構造などの要因で異なってくるが, 多くの予備実験の結果から $\kappa = 4.5$ を採用している.

ここで $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ として,

$$d_{*i}(\boldsymbol{\sigma}) = \min\{d_{1i}(\boldsymbol{\sigma}),\ d_{2i}(\boldsymbol{\sigma}),\ d_{3i}(\boldsymbol{\sigma})\}\ (1 \leq i \leq m), \tag{14}$$

$$d_{*}(\boldsymbol{\sigma}) = \sum_{1 \leq i \leq m} d_{*i}(\boldsymbol{\sigma}). \tag{15}$$

を定義する. そのとき各 $(i = 1, 2, \ldots, m)$ に対して, $d_{*i}(\boldsymbol{\sigma})$ は $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の何れかの計算方法を用いて $B_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の回数の最小重み付き数を示している. そして, それらの合計 $d_*(\boldsymbol{\sigma})$ は行列 B の全ての要素を (10) の順序で計算したときの乗算の回数の最小重み付き数を示している.

$d_*(\boldsymbol{\sigma})$ の数は $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ に依存する, つまり $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の式を B の計算のために適用する以前に. どのようにして索引 (index : $1, 2, \ldots, m$) の順列を決定するかに依存している. そこで,$d_*(\boldsymbol{\sigma})$ を最小にするような順列 $\boldsymbol{\sigma}$ を選択するのが望ましい. 以下に示す定理は行列 $A_1, A_2, \ldots, A_m$ の非零要素の数 $f_1, f_2, \ldots, f_m$ が降順となるように索引 (index) を降順に並び替える順列 $\boldsymbol{\sigma}$ を選んだときに $d_*(\boldsymbol{\sigma})$ が最小になることを示している.

定理 1 *(i)* $\boldsymbol{\sigma}^*$ が全ての $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ の中で $d_*(\boldsymbol{\sigma})$ を最小にすることと, 以下の関係が成り立つことは必要十分である.

$$f_{\sigma^*(1)} \geq f_{\sigma^*(2)} \geq \cdots \geq f_{\sigma^*(m)}. \tag{16}$$

(ii) $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ が *(16)* を満たすと仮定すると, そのとき次の条件を満たすような $q_1 \in \{0, 1, 2, \ldots, m\}$ と $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ が存在する.

$$\left.\begin{array}{ll} d_{1i}(\boldsymbol{\sigma}^*) \leq d_{2i}(\boldsymbol{\sigma}^*), d_{1i}(\boldsymbol{\sigma}^*) \leq d_{3i}(\boldsymbol{\sigma}^*) & 0 < i \leq q_1 \text{ のとき}, \\ d_{2i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*), d_{2i}(\boldsymbol{\sigma}^*) \leq d_{3i}(\boldsymbol{\sigma}^*) & q_1 < i \leq q_2 \text{ のとき}, \\ d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*), d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) & q_2 < i \leq m \text{ のとき}. \end{array}\right\} \tag{17}$$

なお証明は [14] を参照のこと. 次に定理 1 に基づいて、$d_*(\boldsymbol{\sigma})$ を最小にする計算方法を提案する.

Combined Formula $\mathcal{F}$-*(κ): (図 1 参照)

Step A: 行列 A_i $(1 \leq i \leq m)$ の非零要素の数 f_i を数える.

Step B: $f_1, f_2, \ldots, f_m$ の数を指標として索引 (index : $1, 2, \ldots, m$) を降順に並び替える (16). 各 $i = 1, 2, \ldots, m$ に対して, $d_{1i}(\boldsymbol{\sigma}^*)$ (11)' と $d_{2i}(\boldsymbol{\sigma}^*)$ (12)' と $d_{3i}(\boldsymbol{\sigma}^*)$ (13)' を計算する. そして (17) を満たすような $q_1 \in \{0, 1, 2, \ldots, m\}$ と $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ を求める.

Step C: 全ての $i \in \{1, 2, \ldots, m\}$ に対して,

- もし $0 < i \leq q_1$ ならば, $\mathcal{F}$-1 を用いる.
- もし $q_1 < i \leq q_2$ ならば, $\mathcal{F}$-2 を用いる.
- 残りの部分, つまり $q_2 < i \leq m$ ならば, $\mathcal{F}$-3 を用いる.

これらの手順を,HRVW/KSH/M 方向と NT 方向を用いた主双対内点法に容易に組み込むことができる.

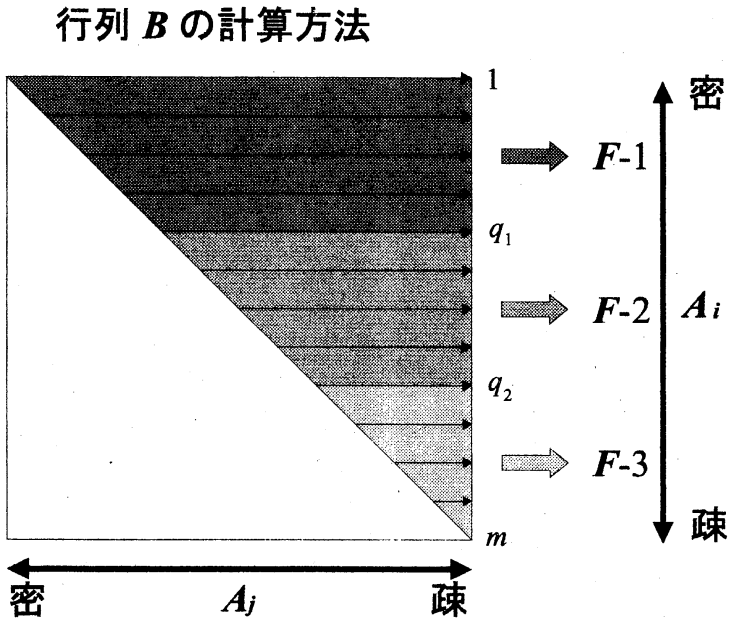

図 1: Combined Formula $\mathcal{F}$-*(κ) を用いた行列 B の計算方法

4 数値実験結果

最後に SDPLIB [4]に収められている問題と構造最適化 [6] の問題を用いて SDPA と他のソフトウェアとの比較実験を行い, SDPA が採用しているデータ構造と計算方法の有効性を検証する. ここで比較実験に用いるソフトウェアは SDPA(Ver5.0), CSDP(Ver 2.3) [5], SeDuMi(Ver 1.02) [6], SDPT3(Ver 2.1) [7] である.

表 1: グラフ分割問題に対する実験結果

	$n = m = 500$	$n = m = 1000$
	CPU(iter)	CPU(iter)
SDPA	282.2(18)	4673.1(23)
CSDP	1127.7(25)	12490.0(32)
SeDuMi	3378.5(30)	26748.6(22)
SDPT3	550.2(17)	7741.3(21)

表 2: 最大クリーク問題に対する実験結果

	$n = 200, m = 1949$	$n = 250, m = 3028$
	CPU(iter)	CPU(iter)
SDPA	520.5(17)	2400.6(18)
CSDP	934.8(23)	3161.0(22)
SeDuMi	822.6(15)	2682.4(14)
SDPT3	798.7(15)	2865.8(15)

表 1から表 4を通して CPU は実行時間 (sec.) を iter は反復回数を示す. また全ての数値実験は DEC Alpha CPU 21164 600MHz (メモリ 1Gbyte) で行なった. 全ての問題で SDPA が最も高速に最適解を産出している. 例えば構造最適化問題 (座屈制約) では (表 4), 行列 A_i

[4]http://www.nmt.edu/ borchers/sdplib.html
[5]http://www.nmt.edu/~borchers/csdp.html
[6]http://www.unimaas.nl/~sturm/software/sedumi.html
[7]http://www.math.nus.sg/~mattohkc/index.html

表 3: 構造最適化問題 (一次固有振動数制約) に対する数値実験結果

	$n = 239, m = 128$	$n = 280, m = 174$
	CPU(iter)	CPU(iter)
SDPA	5.0(20)	5.4(23)
CSDP	26.0(24)	133.2(30)
SeDuMi	33.2(23)	124.6(36)
SDPT3	21.0(15)	22.2(22)

表 4: 構造最適化問題 (座屈制約) に対する数値実験結果

	$n = 719, m = 392$	$n = 1437, m = 792$
	CPU(iter)	CPU(iter)
SDPA	99.4(21)	782.9(22)
CSDP	837.2(28)	7819.4(31)
SeDuMi	1173.8(27)	15441.5(31)
SDPT-3	152.8(20)	956.9(23)

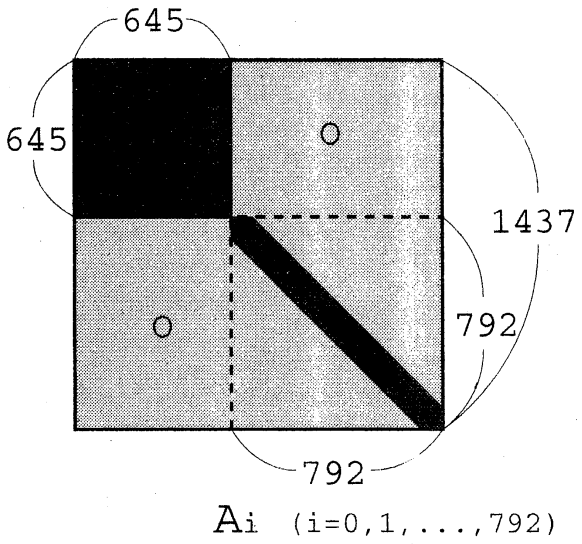

図 2: ブロック対角行列の例 : 構造最適化問題 (座屈制約) $n = 1437, m = 792$

などは, 図 2 のような構造をしている. 第一ブロックは 645×645 の疎行列であり, 第二ブロックは大きさ 792 の対角行列になっている. このようなブロック対角構造を持つ問題では特に SDPA が高速であることが他の多くの数値実験によっても判明しており、解説したデータ構造や計算方法が SDPA 全体の高速化に寄与しているといえる.

謝辞
本研究にあたり貴重な情報や助言等いただきました東京工業大学の小島 (政) 研究室及び松岡研究室の皆さん, また文章を校正していただいた京都大学大学院の寒野善博さんには深く感謝致します.

References

[1] 小島政和: 半正定値計画緩和と大域的最適化; Research Report B-342, Dept. of Mathematical and Computing sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan, July, (1998)

[2] K. Fujisawa, M. Kojima and K. Nakata: SDPA(Semidefinite Programming Algortithm) User's Manual.; Research Report B-308, Dept. of Mathematical and Computing sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan, (1999)

[3] K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata: Numerical evaluation of SDPA (Semidefinite Programming Algorithm); The Proceedings of the Second Workshop on High Performance Optimization Techniques (1999)

[4] 鈴村 豊太郎, 中川 貴之, 松岡 聡, 中田 秀基: クライアント・サーバ型のグローバルコンピューティングシステムの比較 — Ninf, NetSolve, CORBA, Ninf-on-Globus の性能評価 —; 情報処理学会研究会報告 99-HPC-34, (1999)

[5] L. Vandenberghe and S. Boyd: Semidefinite programming; *SIAM Review*, Vol. 38, pp. 49–95 (1996)

[6] M. Ohsaki, K. Fujisawa, N. Katoh and Y. Kanno: Semi-Definite Programming for Topology Optimization of Truss under Multiple Eigenvalue Constraints; to appear in *Comput. Meth. Appl. Mech. Engng.*, (1999)

[7] S. Boyd et al.: Linear Matrix Inequalities in Systems and Control Theory; *SIAM books* (1994)

[8] M. X. Goemans and D. P. Williamson: Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming; *J. ACM*, 42, pp. 1115-1145, (1995)

[9] M. Kojima, S. Shindoh and S. Hara: Interior-point methods for the monotone semidefinite linear complementarity problems; *SIAM J. Optim.*, Vol. 7, pp. 86–125, (1997)

[10] C. Helmberg, F. Rendl, R.J. Vanderbei and H. Wolkowicz: An interior-point method for semidefinite programming; *SIAM J. Optim.*, Vol. 6, No. 2, pp. 342–361, (1996)

[11] R. D. C. Monteiro: Primal-dual path-following algorithms for semidefinite programming; *SIAM J. Optim.*, Vol. 7, No. 3, pp. 663–678, (1997)

[12] M. J. Todd, K. C. Toh and R. H. Tütüncü: On the Nesterov-Todd direction in semidefinite programming; *SIAM J. Optim.*, Vol. 8, No. 3, pp. 769–796, (1998)

[13] S. Mehrotra: On the implementation of a primal-dual interior point method; *SIAM J. Optim.*, Vol 2, pp. 575–601, (1992)

[14] K. Fujisawa, M. Kojima and K. Nakata: Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming; *Mathematical Programming*, Vol. 79, pp. 235–253, (1997)